Runtime Integrity and Presence Verification for Software Agents

Travis Schluessler
Security Architect
Communications Technology Lab
Intel Corporation

Hormuzd Khosravi
Senior Network Software Engineer
Communications Technology Lab
Intel Corporation

Gayathri Nagabhushan
Network Software Engineer
Performance Networking Lab
Intel Corporation

Ravi Sahita
Senior Network Software Engineer
Performance Networking Lab
Intel Corporation

Uday Savagaonkar
Senior Network Software Engineer
Wireless Networking Lab
Intel Corporation

Table of Contents

(Click on page number to jump to sections)

Runtime Integrity and Presence Verification for Software Agents

Travis Schluessler
Security Architect
Communications Technology Lab
Intel Corporation

Hormuzd Khosravi
Senior Network Software Engineer
Communications Technology Lab
Intel Corporation

Gayathri Nagabhushan
Network Software Engineer
Performance Networking Lab
Intel Corporation

Ravi Sahita
Senior Network Software Engineer
Performance Networking Lab
Intel Corporation

Uday Savagaonkar
Senior Network Software Engineer
Wireless Networking Lab
Intel Corporation

Overview: Relief for Network Administrators

Worry is a way of life for many who deal with network security issues. The techniques used by today's intrusion prevention and detection systems (IDSs) are simply not sufficient to protect against the advanced worms and viruses being introduced into today's computing environments.

With 97 percent of U.S. enterprise networks using perimeter firewalls, and 72 percent employing intrusion detection systems, 55 percent of networks (2005 CSI/FBI Computer Crime and Security Survey) were still attacked by worms or viruses, and 95 percent experienced internal system penetration or some kind of unauthorized access. Another report, from the Computer Security Division (CSD), National Institute of Standards and Technology (NIST), showed that 98 percent of platform vulnerability targets in 2004 were software, of which 73 percent of security violations were the result of application vulnerabilities, while 14 percent were the result of operating system (OS) vulnerabilities.

Network administrators may soon get more restful sleep. Many of today's vulnerabilities can be effectively and reliably addressed with a firmware-based approach to integrity services, an approach that has already been prototyped and proven on an IA-32 platform, for both Linux* and Windows*. This firmware-based approach, called System Integrity Services (SIS), makes use of a true, isolated execution environment, as well as sophisticated integrity checks and forgery-protected, execution-presence verification, to detect attacks that circumvent, tamper with, or disable critical software agents running on a host computing system.

Understanding the Enemy

Today's IDSs rely heavily on host-resident software (such as antivirus agents) to monitor and analyze host state. For example, a host software agent can be a device driver, a kernel security agent (such as a firewall), a security service (such as a virtual private network), an operating system (OS) kernel variant, or any other program. Attackers (such as recent variants of the Bagle and Lion worms) try to disable these protected components in the early stages of their attack, so that the attack can proceed undetected. Other attacks try to gain escalated privileges, in order to steal keys that are used for cryptographic operations.

The inherent vulnerability of host-resident software exists because that software is not isolated from other software running within the OS. In fact, all of the server vulnerabilities classified by Microsoft as 'critical' that were exploited in 2003 and 2004 were memory-based, with the software running in memory attacked by worms and viruses trying to circumvent OS protections.

For example, some IDSs are vulnerable to malware (such as the Witty worm) that can acquire unrestricted system access (such as via a rootkit). Many heuristic-based IDSs create too many false positives. Other IDSs rely on externally observed behavior to detect intrusion, and may not be as effective as host-based systems that have access to context. Some IDSs use an isolated execution environment to monitor part or all of the platform, but do not deal with attacks that terminate the execution of memory-resident IDSs or attacks from direct memory-access (DMA) devices. And, most IDSs cannot verify the integrity of user-space components, since those components are not located at a single, fixed, and well-known address.

What is needed is an effective detection and defense system that can deal with many types of intrusions, attacks, and threats, offering reliable integrity checks and unspoofable presence verification.

Designing an Effective Response

Intel researchers have now designed, prototyped, and proven an effective approach to detecting memory-based attacks that cannot be reliably caught with today's host-based IDSs. This approach:

- Reliably detects run-time modification of a monitored software agent.
- Reliably detects when a monitored software agent stops executing.
- Provides a secure, run-time mechanism to protect secrets from other programs.
- Addresses serious threat vectors not dealt with by today's systems, such as attacks from DMA-based devices.
- Is OS-independent, with an isolated execution environment in the monitored host, but which is not accessible from the primary operating system.
- Has a minimal performance overhead.

Intel's approach is a firmware-based, OS-independent prototype called System Integrity Services, or SIS. The approach verifies the integrity and presence of host-resident software agents, and detects attacks that circumvent, tamper with, or disable critical software agents. The firmware-based approach also effectively protects dynamic program data—such as packet filter rules associated with a firewall, or software version information associated with an antivirus application—a critical capability not provided by existing IDSs. And, Intel's approach is resilient against attacks that target the IDS itself.

The approach also offers a generic mechanism to maintain the privacy of secrets used to perform cryptographic operations on the host. This prevents attackers from gaining access to secrets even if they have gained unrestricted access to all system resources, such as via a rootkit. (For detailed information about how SIS protects keys and other sensitive information, refer to the white paper titled "OS-Independent Run-Time System Integrity Services," on the Intel Web site.)

Three Key Concepts

Software agents are made up of a number of critical memory regions that must be verified over the course of its in-memory lifetime to make sure the agent is not modified while running. These memory regions include executable code as well as static and dynamic data. To verify the integrity of these regions and the agent as a whole, Intel's prototype SIS uses a combination of forgery-protected integrity and locality checks.

The SIS is based on three key concepts:

- ***Agent locality:*** The location in memory where a program resides, both its executable code and its static and dynamic data. SIS uses the locality information to determine the identity of a software program that is requesting integrity services. Access to confidential information is restricted, so that only operations executing from the agent's known location in memory (as specified during registration) can make use of such information. Operations attempting to access confidential information from other memory locations are rejected.
- ***Agent integrity:*** Whether an agent has been modified from its original state.
- ***Agent execution state:*** Whether a program is running and being scheduled to run by the OS over time.

When the SIS verifies all three attributes, the system can infer that the agent (excluding unprotected dynamic data) has not been compromised and can be allowed to make use of any agent-specific secrets being protected by the SIS.

System Architecture

The SIS architecture (refer to **Figure 1**) is implemented using the system management mode (SMM) in IA-32 processors as an isolated execution environment. Among its many modules, the SIS includes three key components:

- Integrity measurement manager (IMM)
- Integrity manifest (IM)
- Integrity services module (ISM)

(Detailed information about the architecture and implementation of the SIS prototype is provided in the white paper titled, "OS-Independent Run-Time System Integrity Services.")

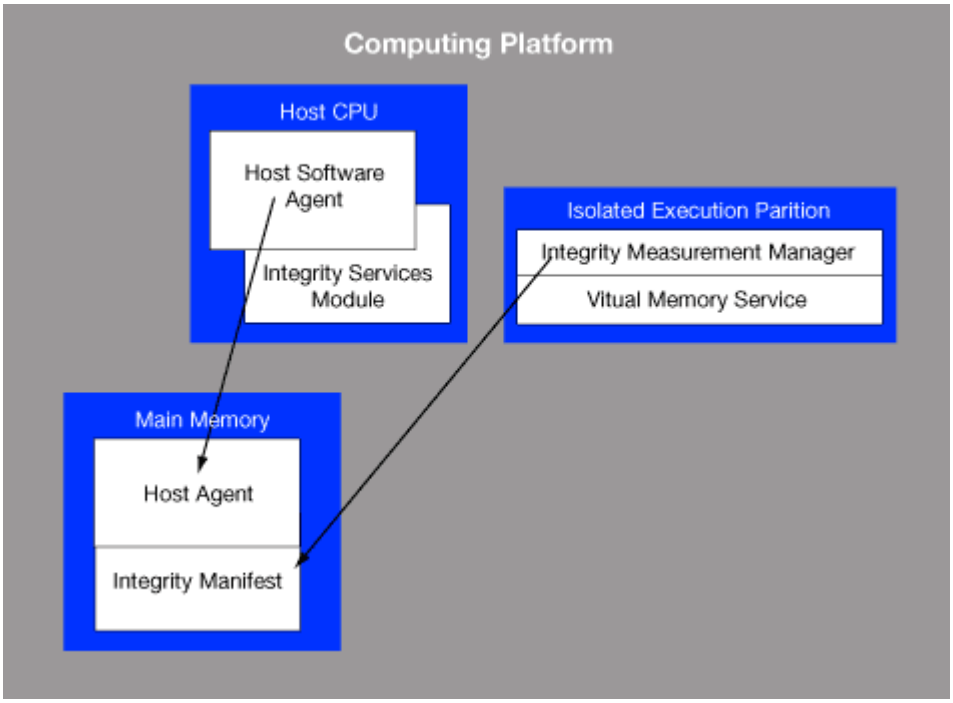

Figure 1. *SIS architecture*

Integrity Measurement Manager

The integrity measurement manager, or IMM, performs critical integrity and execution-detection checks on the protected host agent to make sure the agent has not been tampered with and is still executing. The first integrity check is performed at registration time when the host agent is loaded into host memory. After that, the agent is checked periodically, or on a use-driven basis as it executes.

In order to prove that a host agent is executing, the agent needs to periodically generate an indicator (a "heartbeat") to the IMM that it is alive and executing. To eliminate the inherent vulnerability of these heartbeats, the integrity services module, or ISM, tests the locality of the request for the heartbeat update. Combined with the integrity checks, this prevents an attacker from forging a heartbeat update. In this process, the signature generated by the ISM is returned to the agent, which appends it to the heartbeat message sent to the IMM, which is then verified by the IMM. If an attacker tries to forge the heartbeat message, the forgery is detected due to an incorrect signature, and the IMM can then trigger a policy-based remedial action.

Integrity Manifest

The integrity manifest is used by the ISM to verify the integrity of the host agent. The manifest is the set of information (contents and relocations) that identifies the host agent being protected. Specifically, the manifest is a summary description of the agent's critical components, signed by the program's manufacturer with an asymmetric key, and

verified by the IMM when the agent is loaded into host memory. It also includes a header of offsets and sizes of the section table, relocation table, and symbol table used to identify the agent, as well as other information to verify the integrity of the manifest itself.

Integrity Services Module
The integrity services module, or ISM, performs locality verification to ensure that an agent requesting ISM services is the same agent that registered for those services. The ISM handles agent registration, state verification, updates to protected dynamic application data, etc. The module runs in system management mode (SMM), a privileged execution mode on Intel® x86 processors, and its memory is sequestered from the operating system.

For example, when a system-management interrupt is triggered, a processor enters the system management mode. Once in management mode, the processor executes code and data from a chipset-protected region (SMRAM) of main memory that is inaccessible to the software previously executing on the processor or to software from programmable DMA devices. All interrupts are disabled, and system state information (including register contents) is stored in the system saved-state map region within SMRAM. This state is restored when the processor resumes from the management mode.

Effective Counters to Serious Threats

By verifying the integrity and locality of protected agents, the SIS approach can effectively detect a range of threats not addressed by current methods. For example, once an attacker has breached the kernel boundary to gain the highest levels of privilege on a system, there are an array of methods that can be used to evade detection. These methods include reading and modifying the contents of main memory, directly touching device hardware, modifying scheduler behavior, and so on. The firmware-based SIS deals effectively with these and many other attacks, including:

Image Modification: Attacker modifies an agent's in-memory executable code store to cause undesired behavior (for example, to disable packet filtering in a firewall). ***SIS:*** If agent code is modified in memory, SIS will detect this on subsequent integrity checks.

Disabling the Agent: Attacker disables the agent and prevents its execution. The attacker can kill the process, steal interrupt vectors associated with a program, modify the OS scheduler, and so on. ***SIS:*** If the agent is disabled, the attacker cannot produce the agent's secure heartbeats, and the IMM will detect that valid heartbeats have stopped.

Dynamic Data Modification: Attacker modifies data (such as firewall/filtering rules, or any program state) that should be modified only by the agent. ***SIS:*** Signs and verifies the source and validity of each dynamic data update.

Direct Memory Access (DMA) Device Attack: In a virtualized system, an attacker can instruct an I/O device to perform a direct memory access into memory associated with another virtual machine (or the virtual memory manager), thereby writing an attack payload into the purportedly 'isolated' execution environment. ***SIS:*** Such attacks are detected during the integrity verification process performed by the IMM.

Stealing Cryptographic Secrets: Attacker steals (for example, using rootkit) the cryptographic secrets used by the authorized agent and uses these secrets to impersonate the authorized agent. ***SIS:*** Secrets are stored in SMRAM that is not accessible to even the highest privilege software running in the context of the OS. The integrity services manager uses a combination of entry-point verification and code verification to make sure such secrets are not divulged to unauthorized programs.

Summary

Intel's firmware-based System Integrity Services offers a reliable method for run-time detection of attacks and intrusions that try to tamper with, disable, or circumvent protected software agents. By taking advantage of the capabilities of the system management mode in IA-32 processors, SIS becomes truly independent of the host operating system, and can reliably detect when an attack modifies or halts a protected program.

The SIS approach, effective on its own, complements other platform security solutions, such as software-based intrusion-detection systems and hardware firewalls. It can also be used for encryption, decryption, or any other cryptographic operation that requires secrecy of some part of the data used in the operation.

The SIS architecture is not only practical in detecting many threats currently unaddressed in existing systems, but it consumes a relatively small amount of resources and time. By combining efficient and robust integrity checks with forgery-resistant presence verification, SIS dramatically reduces the number of vectors and available time for a worm to compromise a host agent and propagate through or out of the network.

More Info

For detailed information on SIS, including architecture, implementation (such as agent registration, execution state detection, attack and threat vectors, prototype effectiveness, and so on, refer to the white paper titled, “OS-Independent Run-Time System Integrity Services,” on the Intel Web site. The white paper also includes details about how SIS protects keys and other secrets, verifies packet traffic, handoffs secrets, and so forth.

A related white paper about Intel’s research into an OS-independent worm containment system is titled: "An OS Independent Heuristics-Based Worm-Containment System" [PDF 159KB]. The paper includes an examination of four types of heuristics-based containment systems against well-known, self-propagating worms, and includes information on the pros and cons of various modern approaches to worm containment, test results for various containment systems, and more.

You can also learn much more about Intel's research at the Intel Web site.

Author Bios

Travis Schluessler is a network security researcher at Intel’s Communications Technology Lab. He has an extensive background in platform security, network processors, and ATM networking. Schluessler received his bachelor’s degree in electrical and computer engineering from Carnegie Mellon University.

Hormuzd Khosravi is a senior network software engineer in the Communications Technology Lab at Intel. His current focus is on the integration of Intel’s platform security and manageability technologies with different network architectures and standards. Since he joined Intel in 1999, he has worked on several networking projects including Distributed Control Plane. He was involved in defining and developing industry standards for modular communications platforms in forums such as the IETF ForCES working group and Network Processing Forum. Khosravi received his M.S. in computer engineering from Rutgers University. Prior to joining Intel, Khosravi held the position of research assistant at C&C Research Labs, NEC USA.

Gayathri Nagabhushan is a network software engineer at Intel's Performance Networking Labs within the Communications Technology Lab. Her main focus is on network security initiatives, which include automated worm containment systems, and on providing tamper-resistant platform security solutions for future Intel products. Nagabhushan's current research is focused on developing a security model architecture for verifying both the presence and integrity of host-based security systems and other critical host components. She earned her M.S. in computer science from the University of Madras, and earned a second M.S. in networks and databases in computer science from Portland State University.

Ravi Sahita is a senior network software engineer at Intel's Performance Networking Labs within the Communications Technology Lab. His primary focus is platform and network security/manageability. He is currently working on platform approaches to address network security issues, such as network software integrity protection and worm propagation countermeasures. Sahita previously worked on policy-based network management standards and the development of the Intel® NetStructure® Policy Manager and the Intel® Common Open Policy Services (COPS) SDK. Sahita is a participating member of the Internet Engineering Task Force (IETF) and the Trusted Computing Group (TCG) and has co-authored several key standards-track Internet drafts in the Resource Allocation Protocol IETF working group. Sahita received a B.E. in computer engineering from the University of Bombay, and an M.S. in computer science from Iowa State University.

Uday Savagaonkar joined Intel as a senior product development engineer in 2002, and currently works as a senior network software engineer in the Wireless Networking Labs within the Communications Technology Lab. His research interests include network security, self-healing networks, and performance/threat analysis of secure architectures. He received a master of technology degree in electrical engineering from the Indian Institute of Technology, Mumbai, India and a Ph.D. in electrical engineering from Purdue University.

—End of Technology@Intel Magazine Article—